IDL to C++11
Language Mapping

making CORBA, DDS, and CCM development easy, safe, and fast

Johnny Willemsen
jwillemsen@remedy.nl
Created end of the '90s

Two different approaches for IDL to C++ were proposed

Selection was done based on formal voting, it was more a political decision than a technical one
Lot of ideas were posted and raised throughout the years

Any change to the C++ mapping would break a lot of code

As a result the IDL to C++ mapping was just impossible to change
What changed now?

- A new revision of the C++ programming language itself was in progress.
- This introduces several new constructs for which a new C++ compiler is mandatory.
- When using the new C++ constructs you can't go back anymore to old C++.
- The new C++ language is now formally standardized as C++11.
IDL to C++11 goals

- A much more simplified mapping than the current C++ mapping
- Use C++11 features to
  - Gain performance
  - Reduce coding errors
  - Prevent memory leaks and double deletes
  - Reuse STL and C++11 features rather than reinventing them
C++11 type mapping

- Map basic types to their C++ counterparts
- IDL (w)string maps to std::(w)string
- IDL sequence maps to std::vector
- IDL array maps to std::array
- IDL fixed maps to CORBA::Fixed
Argument passing for basic types, and reference types:
- in: T
- inout: T&
- out: T&
- return: T

Argument passing for complex types:
- in: const T&
- inout: T&
- out: T&
- return: T
- Struct and union now both map to a C++ class
- Providing constructor(s), copy and move operators
- Set of accessors for each member:

  
  void A (T);
  T A (void) const;
  T& A (void);

  void A (const T&);
  void A (T&&);
  const T& A (void) const;
  T& A (void);
IDL enum is mapped to the new C++11 enum class

// IDL
define Color { red, green, blue };

// C++
define class Color : uint32_t { red, green, blue };

Reference types are used for
- Object references
- Valuetypes
- Typecodes

Strong references behave as std::shared_ptr
Weak references behave as std::weak_ptr
References can only be created using CORBA::make_reference<>
No C++ pointers can be passed to the API
C++11 introduces a new type for a nil pointer: `nullptr_t`

References can be checked for nil by comparing them with `nullptr`

Explicit bool conversion operators for comparison
Beta 1 specification lists CORBA as namespace in a few places

Will be moved to omg idl traits that are independent of the specific middleware being used

Allows meta programming without knowing whether you are using CORBA, DDS, or something else
When defining interface Foo, than `CORBA::object_traits<Foo>::ref_type` (aka `CORBA::object_reference<Foo>`, aka `Foo::_ref_type`) is the object reference type.

- C++ traits are used to determine types, no naming rules to be remembered by the user.
- Narrowing using the narrow method (with CORBA could trigger a remote call)
When defining interface Foo, THAN
CORBA::servant_traits<Foo>::base_type is the base class for the user servant class.

Servant references are available as CORBA::servant_traits<Foo>::ref_type (aka CORBA::servant_reference<Foo>)

C++ traits are used to determine types, no naming rules to be remembered by the user.
// IDL to C++ Language Mapping

CORBA::Object_var obj = orb->string_to_object(ior);
Test::Hello_var hello = Test::Hello::_narrow(obj.in ());

if (!CORBA::is_nil (hello.in ())) {
    CORBA::String_var the_string = hello->get_string ();
    std::cout << "hello->get_string () returned " << the_string.in () << std::endl;
    hello->shutdown ();
}


// IDL to C++11 Language Mapping

CORBA::object_reference<CORBA::Object> obj = orb->string_to_object (ior);
Test::Hello::_ref_type hello = Test::Hello::_narrow (obj);

if (hello) {
    std::cout << "hello->get_string () returned " << hello->get_string () << std::endl;
    hello->shutdown ();
}
// IDL to C++ Language Mapping

class Hello : public virtual POA_Test::Hello
{
public:
    Hello (CORBA::ORB_ptr orb) : orb_ (CORBA::ORB::_duplicate (orb)) {
    }
    virtual char * get_string (void) {
        return CORBA::string_dup ("Hello there!");
    }
    virtual void shutdown (void) {
        this->orb_->shutdown (0);
    }
private:
    CORBA::ORB_var orb_;}

// IDL to C++11 Language Mapping

class Hello : public virtual CORBA::servant_traits<Test::Hello>::base_type
{
public:
    Hello (CORBA::object_reference<CORBA::ORB> orb) : orb_ (orb) {
    }
    virtual std::string get_string (void) {
        return "Hello there!";
    }
    virtual void shutdown (void) {
        this->orb_->shutdown (false);
    }
private:
    CORBA::object_reference<CORBA::ORB> orb_;}
Revised submission got recommended for adoption at the OMG March 2012 meeting

Finalization Task Force (FTF) has started work

IDL to C++11 beta 1 specification available on the OMG website

FTF will deliver a V1.0 specification at the OMG September 2012 meeting
For more information about IDL to C++11 check:

- ORBzone at http://www.orbzone.org
  - Discussion forum
  - Articles
- Osportal at http://osportal.remedy.nl
  - More examples